title: External Calls description: Understanding the key primitives required to build great SUAPPs keywords: - contract - deploy - suave - solidity - http - external - data

Making external calls

If you've followed along from the previous tutorial, you'll have deployed a simple contract with `onchain` and `offchain` functions, and extended it so that you can store a private key from another domain and use it to sign transactions after executing some offchain, confidential computation using SUAVE.

Building unique and powerful Suapps often requires one more key primitive (in addition to understanding on and offchain computations and how to use the confidential store).

Suapps can make arbitrary http requests to other domains, fetch data, and use that in their offchain computation

Let's walk through how to do this, first fetching balance information about USDC on Ethereum L1, and then by making a request to Chat GPT's completion endpoint.

Fetching Balances from Ethereum

:::info

If you have SUAVE running locally, stop it and restart it with another flag we'll need for making external calls:

```
bash suave-geth --suave.dev --suave.eth.external-whitelist='*'
```

:::

Create a new contract in your `src` directory called `ExternalCall.sol`, and implement the same pattern we used for storing private keys to store your RPC endpoint:

```solidity // SPDX-License-Identifier: UNLICENSED pragma solidity ^0.8.8;

import "suave-std/Suapp.sol"; import "suave-std/Context.sol"; import "suave-std/Gateway.sol";

interface ERC20 { function balanceOf(address) external view returns (uint256); }

contract ExternalCall is Suapp { Suave.DataId rpcRecord; string public RPC = "RPC";

```
function updateKeyOnchain(Suave.DataId _rpcRecord) public {
    rpcRecord = _rpcRecord;
}

function registerKeyOffchain() public returns (bytes memory) {
    bytes memory rpcData = Context.confidentialInputs();

    address[] memory peekers = new address[](1);
    peekers[0] = address(this);

    Suave.DataRecord memory record = Suave.newDataRecord(0, peekers, peekers, "rpc_endpoint");
    Suave.confidentialStore(record.id, RPC, rpcData);

    return abi.encodeWithSelector(this.updateKeyOnchain.selector, record.id);
}
~~~

} ```

You'll see that we're also importing a new file called `Gateway.sol`, which is what we'll be using in our offchain function to establish a connection to our stored RPC provider such that we can fetch data from other domains. Let's implement the onchain and offchain functions now:

```solidity event Balance(uint256 value);

function onchain() external payable emitOffchainLogs {}

function offchain(address contractAddr, address account) external returns (bytes memory) { bytes memory rpcData = Suave.confidentialRetrieve(rpcRecord, RPC); string memory endpoint = bytesToString(rpcData);

~~~
Gateway gateway = new Gateway(endpoint, contractAddr);
ERC20 token = ERC20(address(gateway));
uint256 balance = token.balanceOf(account);

emit Balance(balance);

return abi.encodeWithSelector(this.onchain.selector);
```

}

function bytesToString(bytes memory data) internal pure returns (string memory) { uint256 length = data.length; bytes memory chars = new bytes(length);
```

~~~
for(uint i = 0; i < length; i++) {
    chars[i] = data[i];
}

return string(chars);

} ```
~~~

Gateway.sol expects an RPC endpoint, passed in as a string, and a contract addresss to query. If you check the reference implementation you'll see we pass in the Beaconchain Deposit contract and use our new gateway to get all depositors. In this example, we'll pass in the USDC contract on Ethereum and fetch the balance of one of Binance's accounts, because why not?

Follow the by-now familiar pattern of compiling, deploying, and storing your RPC endpoint using our spell tool:

bash forge build bash suave-geth spell deploy ExternalCall.sol:ExternalCall If you built suave-geth from source, you may need to specify the whole path:

bash ./<path_to_suave-geth>/build/bin/suave-geth spell deploy ExternalCall.sol:ExternalCall Set your RPC provider (with the key) in the confidential data store: bash suave-geth spell conf-request --confidential-input https://eth-mainnet.g.alchemy.com/v2/<your_key> <your_new_contract_address> 'registerKeyOffchain()' Call the USDC contract on Ethereum, and query a Binance Exchange account (because why not?): bash suave-geth spell conf-request <your_new_contract_address> 'offchain(address,address)' '(0xA0b86991c6218b 36c1d19D4a2e9Eb0cE3606eB48, 0xDFd5293D8e347dFe59E90eFd55b2956a1343963d)' You should see the USDC balance of that Binance exchange account emitted as a log in your console: bash INFO [04-03|17:03:47.464] Running with local devchain settings INFO [04-03|17:03:47.464] No confidential input provided, using empty string INFO [04-03|17:03:47.464] Contract at address 0xcb632cC0F166712f09107a7587485f980e524fF6 INFO [04-03|17:03:47.464] Sending offchain confidential compute request kettle=0xB5fEAfbDD752ad52Afb7e1bD2E40432A485bBB7F INFO [04-03|17:03:48.101] Hash of the result onchain transaction hash=0x654e8b8ee4d66efbc8d37001b8c29e670b74856fc9e047483884888e9e57b2c9 INFO [04-03|17:03:48.101] Waiting for the transaction to be mined... INFO [04-03|17:03:48.206] Transaction mined status=1 blockNum=5 INFO [04-03|17:03:48.377] Logs emitted in the onchain transaction numLogs=1 INFO [04-03|17:03:48.377] Log emitted name=Balance(uint256) value=88,763,447,202,982

Fetching blockchain data from domains beyond SUAVE is as easy as that! We're incredibly hyped to see what you build with this primitive. However, blockchain balances and other data is not the only kind of data you can fetch and use in your Suapps...

## Using Chat GPT in a smart contract on SUAVE

Create a new contract in your src directory called Chat.sol. We'll follow exactly the same procedure to store our ChatGPT API key confidentially. In our offchain() function, we'll implement the few lines required to talk to GPT from your contract:

~~~
```solidity // SPDX-License-Identifier: UNLICENSED pragma solidity ^0.8.8;

import "suave-std/Suapp.sol"; import "suave-std/Context.sol"; import "suave-std/protocols/ChatGPT.sol";

contract Chat is Suapp { Suave.DataId apiKeyRecord; string public API_KEY = "API_KEY";

function updateKeyOnchain(Suave.DataId _apiKeyRecord) public {
    apiKeyRecord = _apiKeyRecord;
}

function registerKeyOffchain() public returns (bytes memory) {
    bytes memory keyData = Context.confidentialInputs();

    address[] memory peekers = new address[](1);
    peekers[0] = address(this);

    Suave.DataRecord memory record = Suave.newDataRecord(0, peekers, peekers, "api_key");
    Suave.confidentialStore(record.id, API_KEY, keyData);

    return abi.encodeWithSelector(this.updateKeyOnchain.selector, record.id);
}

event Response(string messages);

function onchain() public emitOffchainLogs {}

function offchain() external returns (bytes memory) {
    bytes memory keyData = Suave.confidentialRetrieve(apiKeyRecord, API_KEY);
    string memory apiKey = bytesToString(keyData);
    ChatGPT chatgpt = new ChatGPT(apiKey);

    ChatGPT.Message[] memory messages = new ChatGPT.Message[](1);
    messages[0] = ChatGPT.Message(ChatGPT.Role.User, "Say hello world");

    string memory data = chatgpt.complete(messages);

    emit Response(data);

    return abi.encodeWithSelector(this.onchain.selector);
}

function bytesToString(bytes memory data) internal pure returns (string memory) {
    uint256 length = data.length;
    bytes memory chars = new bytes(length);

    for(uint i = 0; i < length; i++) {
        chars[i] = data[i];
    }

    return string(chars);
}
~~~

} ```

If you look under the hood at the ChatGPT.sol file you are importing, you will see it uses the general-purpose Suave.doHttpRequest() precompile to achieve the magic of querying an LLM from within an otherwise-ordinary smart contract.

Recompile, deploy, and call the offchain function using our spell tool:

bash forge build bash suave-geth spell deploy Chat.sol:Chat bash suave-geth spell conf-request --confidential-input <your_api_key> <your_new_contract_address> 'registerKeyOffchain()' Once your API key is set in the confidential store, you can send any prompt you like, passing it in as an argument to your offchain function like this: bash suave-geth spell conf-request <your_new_contract_address> 'offchain()'

You should see the response from ChatGPT printed in your console:

```
bash INFO [04-03|20:40:16.752] Running with local devchain settings INFO [04-03|20:40:16.753] No confidential input provided, using empty string INFO [04-03|20:40:16.753] Contract at address 0x975235826142D438d91c0a36171325eF75d7047b INFO [04-03|20:40:16.753] Sending offchain confidential compute request kettle=0xB5fEAfbDD752ad52Afb7e1bD2E40432A485bBB7F INFO [04-03|20:40:18.112] Hash of the result onchain transaction hash=0x70e9ab5f00d78a4111a61d0732478de2031b73a8e558f41a0eac42c3dc0aee1c INFO [04-03|20:40:18.112] Waiting for the transaction to be mined... INFO [04-03|20:40:18.215] Transaction mined status=1 blockNum=25 INFO [04-03|20:40:18.352] Logs emitted in the onchain transaction numLogs=1 INFO [04-03|20:40:18.352] Log emitted name=Response(string) messages="Hello, world!"
```

Congratulations! You now have all the tools you need to build powerful and unique Suapps which simply are not possible other public blockchains.

Foundry

One additional note: if you wish to extend this work and integrate it into a Foundry project of your own, please make sure to adjust the foundry.toml file to include the following (which is the equivalent of running suave-geth with the --suave.eth.external-whitelist='*' flag):

```
bash [profile.suave] whitelist = ["*"]
```

Wrapping Up

:::info

If you'd like to understand how the spell tool we have been using throughout these tutorials ServiceWorkerRegistration, you can read the **Golang source code here**.

:::

Feel free to post any Suapp you build on our forum and we'll be happy to help you review and iterate, as well as invite you to our Developer Chat so you can see what others are building and share tips and best practices with the sharpest engineers we know.